



Hidden costs of DIY internal tools: Why low-code is the future



Hidden costs of DIY internal tools

And why low-code is the future

Introduction: Why internal tools matter

Every business reaches a point where off-the-shelf tools or spreadsheets just can't keep up with their needs. That's when the decision to build internal tools in-house feels logical—after all, who knows your processes better than your own team? But what looks like a cost-saving, flexible solution on the surface often turns into a hidden drain on resources and productivity.

DIY internal tools seem appealing initially. They offer control and customization. But as the business grows, maintaining these tools becomes a significant burden. The result? Engineers bogged down in **maintenance**, **technical debt** piling up, and operational **bottlenecks** that slow down progress.

In this eBook, we'll uncover the unseen challenges of building and maintaining DIY internal tools. We'll explore why the **traditional approach** might not be as cost-effective as it seems and why low-code platforms are emerging as the smarter alternative for creating scalable, secure, and efficient internal tools.

 Authored by



Gayatri Sachdeva
Developer Engagement Strategist
 DronaHQ



Roger Jipp
Director of Sales
 DronaHQ

Table of content

Introduction: Why Internal Tools Matter

Chapter 1: The DIY Trap	4
Development Costs and Time	4
Technical Debt: The Long-Term Consequences	4
Lackluster User Experience: A Common Flaw	5
Key-Person Dependency: A Serious Risk	5
Scaling Challenges: When DIY No Longer Fits	6
Chapter 2: Why Low-Code is the Solution to DIY Challenges	8
What is Low-Code?	8
Speeding Up Development Without Sacrificing Quality	9
Customization and Flexibility for Engineering Teams	9
Scaling Internal Tools with Low-Code	10
Chapter 3: Breaking Down the Costs – DIY vs Low-Code	12
Development Costs: Comparing DIY and Low-Code	12
Maintenance and Technical Debt: Long-Term Implications	13
Infrastructure and DevOps Costs: Overlooked Expenses	14
Product Management and Iteration Costs: Ongoing Investment	14
Chapter 4: Security & Compliance – Mitigating Risks with Low-Code	16
Built-In Security Features	16
Compliance Management Out of the Box	17
Automatic Updates and Continuous Security Monitoring	17
Auditing and Reporting for Compliance	18
Chapter 5: Empowering Engineering Teams for Strategic Impact	20
Engineers as Strategic Leaders, Not Just Coders	20
Collaboration Without the Compromise of Control	22
Chapter 6: Future of Internal Tools – Securing Operational Leverage with Low-Code	24

Chapter 1: The DIY Trap

Building internal tools from scratch seems logical at first. You know exactly what your business needs, and having control over the development process feels empowering. But as with many things, the hidden complexities of DIY internal tools can outweigh the benefits, leading to costly inefficiencies down the road.

Developing DIY internal tools demands a team of frontend and backend developers, infrastructure and DevOps specialists, product managers, and designers. Even a basic team can cost a small startup over \$1 million annually, with costs escalating to tens of millions for mid-sized companies and over \$100 million for global enterprises. Both those are not the only costs. Here's what most teams don't see coming.

Stat Alert: Engineers spend up to 40% of their time on tasks unrelated to core product development (McKinsey).

1. Development time: A silent drain on resources

Time is money, and nowhere is this more evident than in custom tool development. While the initial build of a DIY tool may seem manageable, it's the **ongoing updates, fixes, and feature requests** that quickly turn into a bottleneck. **Every tweak to the tool requires engineers to pause critical work on core products, pulling focus and delaying innovation.**

Key takeaway: Every hour spent on DIY tool maintenance is an hour not spent on building your product or improving customer experience.



2. Technical debt: The long-term consequence

DIY internal tools, by nature, **grow over time**. What starts as a small solution to a simple problem often balloons into a sprawling, complex system as more features are added. Without proper planning and scalability in mind, this results in significant technical debt—quick fixes, patched-up solutions, and outdated frameworks. Soon enough, your once-pristine tool becomes a burden, requiring extensive refactoring to keep it running smoothly.

Key takeaway: The more a DIY internal tool is patched and expanded, the greater the cost to maintain and refactor it in the future.



3. Security and compliance: **Not optional, but essential**

DIY tools are often built with functionality in mind, not security. This leaves **gaps in role-based access control**, data encryption, and **audit logs**—issues that can lead to compliance violations and security breaches.

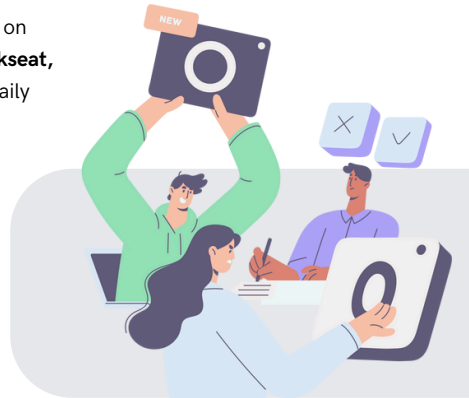
Building a secure, **compliant internal tool from scratch requires expertise, ongoing vigilance, and significant resources**. Many businesses underestimate the cost and complexity of ensuring robust security in their DIY solutions.

Key takeaway: Security failures in DIY tools can expose your business to significant risk, both in terms of compliance and reputation.

4. Lackluster user experience: **A common flaw**

In many cases, internal tools are built without a focus on user experience. **Front-end design often takes a backseat, leading to clunky, unintuitive interfaces** that make daily tasks harder for teams. Without a dedicated focus on UX, these tools may lack the polish necessary to deliver a seamless experience.

Key takeaway: The lack of attention to front-end design in DIY tools results in inefficient, frustrating user experiences.



5. Key-person dependency: **A serious risk**

When DIY tools are built and maintained by a small team—or worse, a single engineer—it creates a critical dependency. If that person leaves, their knowledge goes with them. Training new engineers on the tool becomes costly and time-consuming, slowing down your operations and leaving you vulnerable.

Key takeaway: Relying on a single person or a small team to maintain custom-built tools can lead to major disruptions if they depart.

6. Scaling challenges: **When DIY no longer fits**

As business grows, so does the complexity of internal tools. Scaling these tools to accommodate larger teams and higher data volumes requires constant attention and resources; keep up when user demands increase, leading to performance issues, integration failures, and data bottlenecks.

Key takeaway: Relying on a single person or a small team to maintain custom-built tools can lead to major disruptions if they depart.

Stat Alert: Firms that scale prematurely without the right internal systems in place fail 70% of the time (Startup Genome).

Chapter 1 Recap – Key Takeaways

1. **Budget constraints:** Engineering time across teams is expensive and stretches resources thin.
2. **Development time:** Every hour spent on DIY tools pulls engineers away from core product work.
3. **Technical debt:** Constant patches and growth lead to long-term maintenance burdens.
4. **Innovation lag:** As the tool evolves, future improvements become slower and harder to implement.
5. **Security risks:** Application sprawl creates vulnerabilities due to a lack of standardized IT governance.
6. **Key person risk:** Dependency on single developers leads to major risks when they leave.
7. **Lackluster user experience:** DIY tools often lack the frontend design expertise needed for smooth user experiences.
8. **Scaling challenges:** DIY tools struggle to keep up with growing teams and changing requirements.
9. **Maintenance demands:** DIY tools eventually become bottlenecks, requiring constant attention and pulling resources away from innovation.

Chapter 2: Why Low-Code is the Solution to DIY Challenges

In the previous chapter, we explored how DIY internal tools come with hidden costs—extended development times, technical debt, scalability issues, and security concerns. These challenges slow down engineering teams and keep them tied up in maintenance instead of innovation.

Low-code platforms offer a solution to these problems. By providing pre-built components, automation tools, and flexible integrations, low-code allows engineering teams to build and scale internal tools much faster, while addressing the hidden costs associated with traditional DIY development.

In this chapter, we'll dive into what low-code really is and how it addresses the specific challenges we discussed earlier.

Section 1:

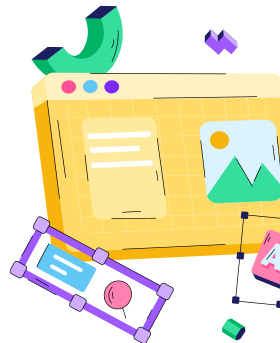
What is low-code and how does it address DIY challenges?

Low-code development platforms allow engineers to build applications using pre-built components and a visual interface but with the flexibility to add custom code where needed. Unlike DIY solutions, which require custom development from scratch, low-code provides a way to streamline development while **maintaining control over the outcome**.

Stat Alert: By 2024, 65% of all app development will be done using low-code platforms (Gartner).

How low-code solves DIY challenges:

- **Faster development:** Low-code platforms reduce the time required to build internal tools by offering ready-made components.
- **Simplified scalability:** With built-in support for handling larger data sets and user loads, low-code tools are built to scale alongside your organization, avoiding the need for constant redevelopment.
- **Security from the start:** Low-code platforms come with pre-configured security features like role-based access control (RBAC) and encryption, reducing the need for engineers to build and manage security protocols from scratch.



Section 2:

Speeding up development without sacrificing quality

A key reason engineering teams benefit from low-code is the ability to accelerate the development process. Low-code platforms let engineers focus on business logic and integration while handling the repetitive groundwork like UI building or data binding through pre-built elements. These tools help engineers move faster without sacrificing quality.

How low-code accelerates development:

- **Pre-built components:** Ready-to-use components such as forms, buttons, and data tables reduce the time spent on building user interfaces.
- **Faster prototyping:** Engineers can quickly create functional prototypes using drag-and-drop tools, gather feedback from stakeholders, and iterate faster.
- **Pre-configured integrations:** Built-in connectors make it easier to integrate with databases, APIs, and other internal systems without extensive configuration, reducing engineering overhead.

Stat Alert: Low-code development can reduce development time by up to 80%, with some businesses reporting a 300% return on investment (ROI) (Forrester).

Section 3:

Customization and flexibility for engineering teams

One concern many engineering teams have about adopting low-code platforms is the potential lack of control over customization. However, low-code platforms are designed to balance speed with flexibility, allowing engineers to write custom code and extend platform capabilities as needed.

How low-code platforms offer flexibility:

- **Custom logic:** Engineers can easily insert custom code (JavaScript, SQL, etc.) into their applications to handle unique business requirements or workflows.
- **Custom APIs:** In addition to the platform's pre-built connectors, engineers can integrate with proprietary systems or third-party APIs by writing their own API connections.
- **Flexible components:** Engineers can take pre-built components and modify them to fit specific use cases, ensuring that the final tool is tailored to the business's needs.

Section 4:

Scaling internal tools with low-code

DIY internal tools often struggle to scale as businesses grow, requiring constant rework to handle more data or users. Low-code platforms, by contrast, are built with scalability in mind, allowing engineering teams to focus on developing tools without worrying about infrastructure limits.

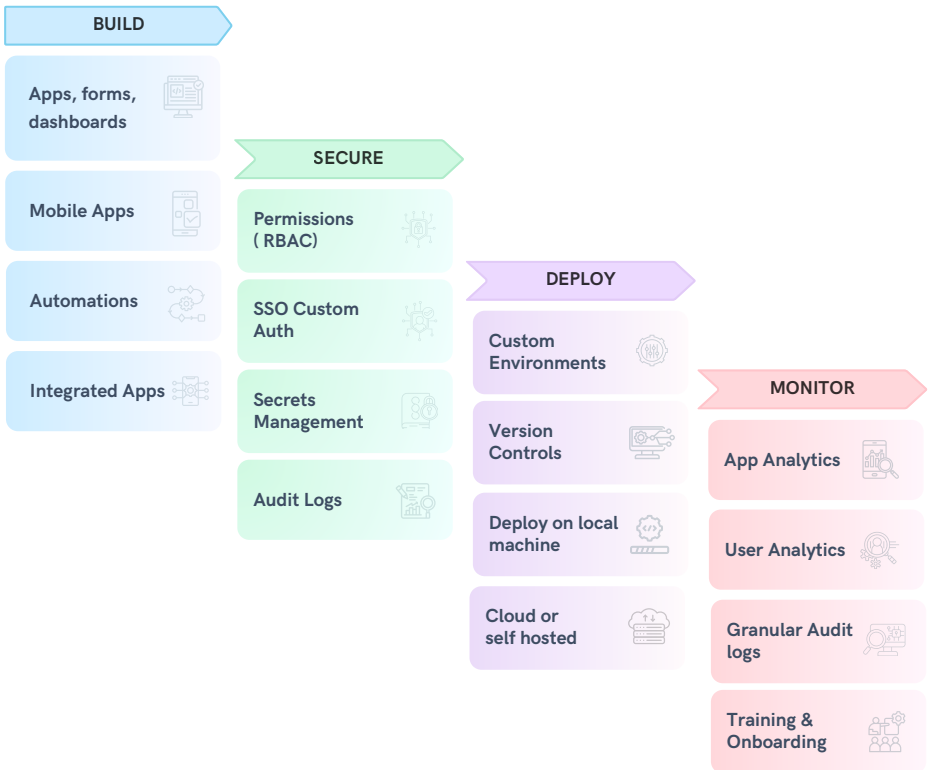
How low-code ensures scalability:

- **Automatic scaling:** Cloud-based low-code platforms handle increased user loads and data requirements without the need for manual intervention from engineers.
- **Performance optimization:** Built-in performance monitoring tools allow engineers to identify and resolve bottlenecks, ensuring applications run efficiently as they scale.
- **Seamless updates:** Engineers can extend or modify internal tools as needed, without the need to refactor entire systems or rebuild from scratch.

Chapter 2 recap – key takeaways:

1. Low-code platforms reduce development time by providing pre-built components and handling repetitive tasks like UI building.
2. Scalability is built-in, meaning internal tools can grow alongside the business without significant redevelopment efforts.
3. Customization remains intact: Engineers retain full control over the logic and integrations, allowing them to meet unique business requirements.
4. Security is built-in: Pre-configured security features reduce the need for engineers to develop and maintain custom security protocols.

Most comprehensive low-code platform to build internal tools



Chapter 3: Breaking Down the Costs – DIY vs Low-Code

Building and maintaining internal tools comes with significant costs, many of which aren't always apparent at the start. Engineering teams often get bogged down in development, maintenance, and scalability issues when handling DIY internal tools. Low-code platforms, by contrast, offer a more efficient and cost-effective approach.

In this chapter, we will break down the hidden costs of DIY internal tools and compare them to the benefits of low-code. We'll explore how low-code reduces the burden on engineering teams, allowing them to focus on innovation rather than firefighting.

1. Development Costs

DIY / Traditional Approach

Building internal tools from scratch requires engineers to invest substantial time and effort into creating every component, from the UI to backend integrations. This can take weeks or months, delaying development and consuming valuable engineering time.

- **High development time:** Custom development means building everything from scratch, which takes significant time.
- **Opportunity cost:** Every hour spent on internal tools is time engineers are not spending on core product development.

Low-Code Approach

Low-code platforms come with pre-built components and ready-made integrations, drastically reducing the time required to build internal tools. Engineers can quickly assemble tools, allowing them to focus on higher-level innovation.

- **Time savings:** Pre-built components reduce development time, allowing engineers to deliver tools faster.
- **Focus on innovation:** Engineers can focus on more strategic tasks, freeing them from repetitive development.

Stat Alert: 42% of startups fail because of operational inefficiencies (Startup Genome)

2. Security and Compliance Costs

DIY / Traditional Approach

Securing DIY tools requires engineers to build custom security features such as encryption, role-based access control (RBAC), and audit logs. Additionally, ensuring compliance with regulations like GDPR or SOC2 is complex and resource-intensive.

- **Custom security requirements:** Engineers must develop security protocols from scratch, consuming time and resources.
- **Compliance burden:** DIY tools may not meet evolving regulatory requirements, increasing the risk of non-compliance.

Low-Code Approach

Low-code platforms come with built-in security features like RBAC and encryption, ensuring that internal tools are secure from the start. Many low-code platforms are also designed to meet common compliance standards, reducing the burden on engineers.

- **Built-in security:** Enterprise-grade security features are provided out of the box.
- **Compliance-ready:** Platforms are often designed to meet industry standards like GDPR, HIPAA, and SOC2.

3. Maintenance and Technical Debt

DIY / Traditional Approach

Once built, DIY internal tools require ongoing maintenance. Bugs, performance issues, and feature updates demand continual attention. Over time, DIY tools accumulate technical debt, becoming harder to update and maintain.

- **Ongoing maintenance:** Engineers must continuously update and maintain DIY tools.
- **Technical debt:** As the tools age, they become more difficult to modify and maintain.

Low-Code Approach

Low-code platforms automatically handle updates and scalability, reducing the need for manual maintenance. Pre-built components and templates minimize technical debt, allowing tools to evolve more easily.

- **Reduced maintenance:** Platforms manage updates and scaling, reducing the maintenance burden on engineers.
- **Minimized technical debt:** Pre-built components ensure that tools remain easy to modify and extend.

Stat Alert: Engineers spend up to 40% of their time on tasks unrelated to core product development (McKinsey).

4. Infrastructure and DevOps Costs

DIY / Traditional Approach

Securing DIY tools requires engineers to build custom security features such as encryption, role-based access control (RBAC), and audit logs. Additionally, ensuring compliance with regulations like GDPR or SOC2 is complex and resource-intensive.

- **Custom security requirements:** Engineers must develop security protocols from scratch, consuming time and resources.
- **Compliance burden:** DIY tools may not meet evolving regulatory requirements, increasing the risk of non-compliance.

Low-Code Approach

Low-code platforms come with built-in security features like RBAC and encryption, ensuring that internal tools are secure from the start. Many low-code platforms are also designed to meet common compliance standards, reducing the burden on engineers.

- **Built-in security:** Enterprise-grade security features are provided out of the box.
- **Compliance-ready:** Platforms are often designed to meet industry standards like GDPR, HIPAA, and SOC2.

5. Product Management and Iteration Costs

DIY / Traditional Approach

Once built, DIY internal tools require ongoing maintenance. Bugs, performance issues, and feature updates demand continual attention. Over time, DIY tools accumulate technical debt, becoming harder to update and maintain.

- **Ongoing maintenance:** Engineers must continuously update and maintain DIY tools.
- **Technical debt:** As the tools age, they become more difficult to modify and maintain.

Low-Code Approach

Low-code platforms automatically handle updates and scalability, reducing the need for manual maintenance. Pre-built components and templates minimize technical debt, allowing tools to evolve more easily.

- **Reduced maintenance:** Platforms manage updates and scaling, reducing the maintenance burden on engineers.
- **Minimized technical debt:** Pre-built components ensure that tools remain easy to modify and extend.

Chapter 3 recap – key takeaways:

- Low-code platforms reduce development time by providing pre-built components and handling repetitive tasks like UI building.
- Scalability is built-in, meaning internal tools can grow alongside the business without significant redevelopment efforts.
- Customization remains intact: Engineers retain full control over the logic and integrations, allowing them to meet unique business requirements.
- Security is built-in: Pre-configured security features reduce the need for engineers to develop and maintain custom security protocols.

Chapter 4: Security & Compliance – Mitigating Risks with Low-Code

One of the most critical considerations when building internal tools is ensuring that they are secure and compliant with industry regulations. In-house development often requires engineering teams to manually implement security protocols and stay up to date with evolving compliance standards, which can quickly become a resource-intensive burden.

Low-code platforms simplify this process by providing built-in security features and compliance tools. These platforms offer everything from encryption to role-based access controls (RBAC) out of the box, ensuring that internal tools remain secure while meeting industry-specific regulatory requirements like GDPR, SOC2, and HIPAA.

Stat Alert: Low-code platforms often come with built-in compliance tools like GDPR, HIPAA, and SOC2 compliance.

1. Built-In Security Features

Low-code platforms are designed with security in mind from the ground up. This means that rather than building custom security features for each tool, engineering teams can leverage pre-configured, enterprise-grade security protocols that protect data and ensure user privacy.

Key Security Features:

- **Role-Based Access Control (RBAC):** Low-code platforms allow you to define user roles and assign permissions accordingly. This ensures that users only have access to the data and features relevant to their role, reducing the risk of unauthorized access.
- **Data Encryption:** Both data at rest and in transit are automatically encrypted, ensuring that sensitive business information is protected from unauthorized access or interception.
- **Authentication and Single Sign-On (SSO):** Low-code platforms provide built-in authentication methods, including SSO, which integrates with your existing identity management systems to streamline login and ensure secure user authentication.
- **Audit Logs and Monitoring:** Platforms include detailed audit logs that track user actions, enabling teams to monitor for suspicious behavior, investigate security incidents, and meet compliance requirements.



2. Compliance Management Out of the Box

Maintaining compliance with industry regulations is a major concern for companies handling sensitive data, especially in sectors like healthcare, finance, and retail. DIY internal tools require engineering teams to manually configure compliance features, which can be complex and time-consuming. Low-code platforms simplify this by offering compliance certifications and tools that ensure your internal tools meet industry standards without needing to build custom solutions.

Regulatory Compliance Made Easy

- **GDPR:** Low-code platforms offer built-in features to ensure compliance with the GDPR. These include data subject rights management, audit logs, and consent tracking.
- **HIPAA:** For companies in highly regulated industries such as healthcare, low-code platforms provide HIPAA-compliant infrastructure, including encryption and access controls that meet specific security standards. Platforms that support self-hosting further make sure you are HIPAA compliant.
- **SOC2:** Platforms often include SOC2 certifications, ensuring that internal tools meet rigorous security, availability, processing integrity, confidentiality, and privacy requirements.



3. Automatic Updates and Continuous Security Monitoring

Security threats evolve over time, and so do compliance regulations. DIY internal tools require constant updates to stay secure and compliant, placing a heavy burden on engineering teams. Low-code platforms address this issue by offering automatic updates and continuous security monitoring, ensuring that internal tools remain secure without the need for manual intervention.

Continuous Updates:

- **Security patches:** Low-code platforms regularly release security patches and updates to keep tools protected from the latest threats.
- **Compliance adjustments:** As regulations change, low-code platforms automatically update compliance features to ensure that internal tools remain compliant with evolving standards.
- **Zero downtime:** Platforms handle these updates seamlessly in the background, ensuring that internal tools experience minimal downtime during maintenance or updates.

4. Auditing and Reporting for Compliance

Auditing is an essential aspect of maintaining compliance, particularly in regulated industries like healthcare and finance. With DIY tools, building audit trails and generating compliance reports can be complex and resource-intensive. Low-code platforms simplify this process by offering built-in auditing and reporting features, allowing businesses to meet compliance requirements without additional development work.



Audit Trails:

- **Automated logging:** Low-code platforms automatically generate audit logs that track user actions, data access, and changes to the system. These logs can be easily accessed and reviewed for compliance audits.
- **Comprehensive reports:** Platforms provide customizable reporting features that allow engineering and compliance teams to generate reports for internal review or external audits with minimal effort.
- **Data retention policies:** Low-code platforms support data retention policies that ensure audit logs are kept for the required duration, helping businesses meet regulatory retention requirements.

5. Security and Compliance at Scale

As businesses grow and scale, security and compliance requirements become even more critical. DIY tools often struggle to keep up with increasing security demands, particularly as user bases grow and tools become more complex. Low-code platforms are built to scale, offering flexible and secure environments that meet the needs of growing businesses.

Scalability and Security:

- **Scalable security features:** As the user base grows, low-code platforms ensure that security features such as encryption, authentication, and RBAC scale automatically, without requiring manual rework by engineering teams.
- **Cross-industry compliance:** Whether your business is in healthcare, finance, retail, or another regulated industry, low-code platforms offer cross-industry compliance tools that grow with your business.

Chapter 4 recap – key takeaways:

- **Built-in security features:** Low-code platforms provide pre-configured security protocols such as encryption, RBAC, and audit logging, significantly reducing the burden on engineering teams.
- **Compliance made easy:** Low-code platforms offer built-in compliance tools for regulations like GDPR, HIPAA, SOC2, and PCI-DSS, ensuring that internal tools meet industry standards without manual configuration.
- **Automatic updates:** Platforms handle regular security updates and compliance adjustments automatically, reducing the risk of vulnerabilities and ensuring ongoing compliance.
- **Audit trails and reporting:** Built-in audit logs and customizable reporting features simplify the compliance process, making it easier to meet regulatory requirements and pass external audits.
- **Scalability:** Low-code platforms are designed to scale security and compliance features automatically, ensuring that growing businesses remain secure and compliant without additional engineering work.

Chapter 5: Empowering Engineering Teams for Strategic Impact

Low-code platforms don't just make the development process faster—they fundamentally shift how engineering teams operate, from managing everyday tasks to driving strategic, high-value innovations. With fewer mundane responsibilities like infrastructure management and routine updates, engineers are free to focus on projects that drive business outcomes, helping the team take on ambitious goals.

This chapter will explore how low-code empowers engineers to go beyond solving small issues and start becoming change-makers within their organizations.

1. Engineers as Strategic Leaders, Not Just Coders

Low-code transforms engineers from being implementers of business requirements to strategic decision-makers who proactively solve problems. Instead of getting bogged down by daily operational tasks, engineers can now lead initiatives that align directly with business objectives.

Proactive Problem Solving

- **Engineering-led innovation:** Engineers can now initiate solutions to business challenges rather than just responding to requests.
- **Developing custom-fit solutions:** Low-code enables engineers to prototype, iterate, and deliver systems that are tailored for specific business operations, driving more value for the organization.



2. Harnessing the Power of Rapid Experimentation



Low-code platforms eliminate the technical hurdles involved in experimenting with new tools and systems. Engineers can now conduct multiple experiments simultaneously, testing different approaches to solving business problems and improving internal processes.

Agile Testing of Ideas

- **Instant iterations:** By shortening the development cycle, engineers can test out new hypotheses and deploy working versions in real time.
- **Building smarter systems:** With rapid iteration cycles, engineers can use low-code to **introduce data-driven models, AI,** or automation into workflows, bringing intelligent solutions to the business.

Stat Alert: Firms that adopt automation early scale 30% faster than those that rely on manual processes (Bain & Company).

3. Engineers as Enablers of Data-Driven Insights

Low-code allows engineers to integrate advanced data analytics and machine learning into internal tools, transforming department decision-making processes. Engineers can create dashboards that surface actionable insights from business data, leading to smarter decisions.

Driving Business Outcomes Through Data

- **Advanced analytics:** Engineers can rapidly build tools that analyze internal data streams, helping departments make more informed, data-backed decisions.
- **AI/ML integration:** Low-code allows engineers to integrate machine learning algorithms into internal tools to automate predictions, improve customer service, or optimize operations.



4. Collaboration Without the Compromise of Control

While collaboration is a key benefit of low-code, the chapter should emphasize that engineers **don't lose control** over critical aspects of development. Engineers can lead the architectural design while allowing non-technical teams to make small adjustments within a controlled environment, maintaining oversight.

Managing Collaboration Effectively

- **Controlled autonomy:** Non-technical users can configure and modify internal tools within set parameters, while engineers maintain overall control of the system architecture and integrations.
- **Co-designing without silos:** Engineers lead the technical architecture while enabling teams to collaborate on operational tweaks, improving both communication and delivery speed.



5. Long-Term Impact

By building systems that are scalable, flexible, and intelligent, engineers can use low-code to deliver transformational change, from optimizing operations to creating new customer experiences.

Transformational Projects

- **Building for the future:** With fewer constraints on development time, engineers can focus on long-term projects that have the potential to transform entire business units.
- **Scaling innovation:** Low-code makes it possible to test, deploy, and scale new solutions rapidly, helping businesses stay agile in competitive markets.

Chapter 5 recap – key takeaways:

- Engineers as strategic leaders: Low-code empowers engineers to move from tactical execution to leading innovation, contributing to business growth.
- Rapid experimentation drives smarter systems: Engineers can quickly test new approaches, incorporating AI and data-driven insights into business solutions.
- Collaboration without losing control: Engineers maintain control over the architecture while enabling non-technical teams to contribute to internal tools, fostering effective cross-team collaboration.
- Long-term impact: Low-code enables engineers to drive large-scale, transformational innovations that have lasting impacts on the business.

Chapter 6: Future of Internal Tools – Securing Operational Leverage with Low-Code

The strategic importance of enhancing developer productivity and end-user efficiency cannot be overstated. Investing in operational software to automate business processes is not just a choice but a necessity, particularly in a challenging macroeconomic environment. Low-code platforms are at the forefront of this transformation, driving down costs while significantly improving customer experiences.

The Logical Path Forward: Low-Code Platforms

Low-code technology has proven to be a game-changer, allowing businesses to find operating leverage by streamlining development processes and boosting the efficiency of business users. By reducing the reliance on extensive coding and leveraging visual development environments, low-code platforms minimize the time and resources traditionally required for software development.

Statistical Evidence Supporting Low-Code Adoption:

- The low-code market is expected to grow from \$13 billion in 2020 to over \$187 billion by 2030, reflecting a compounded annual growth rate of 26.1% (Source: MarketsandMarkets).
- A study by Forrester found that low-code development can reduce development time by up to 80%, with some businesses reporting a 300% return on investment (ROI) due to decreased development costs and quicker time-to-market.
- According to Gartner, by 2024, 65% of all app development functions will be performed by low-code platforms, up from less than a third today.

Driving Down Costs While Enhancing Customer Experience

Low-code platforms encapsulate the core value proposition required in today's business landscape:

- Engineering Time Savings: They significantly cut down on the hours needed to build and maintain internal tools, allowing engineering teams to focus on high-value projects.
- Exponential Efficiency for Business Users: By enabling non-technical team members to manage updates and integrations, low-code platforms empower a broader segment of the workforce, enhancing overall productivity.

Embrace Low-Code Today

The future of internal tools is clearly leaning towards low-code solutions. As businesses strive to remain competitive in a rapidly evolving market, adopting low-code technology is a strategic decision that yields substantial operational advantages.

To learn more visit www.dronahq.com or [book a 1:1 demo](#) with DronaHQ low-code platform Solution Engineers to help you with your internal tooling.



Build custom apps, incredibly fast